Introduction to Parallel and Distributed Computing Exercise 3 (May 30)

Wolfgang Schreiner
Wolfgang.Schreiner@risc.uni-linz.ac.at

The result is to be submitted by the deadline stated above via the Moodle interface as a .zip or .tgz file which contains

- a PDF file with
  - a cover page with the title of the course, your name, Matrikelnummer, and email-address,
  - the source code of the sequential program,
  - the demonstration of a sample solution of the program,
  - the source code of the parallel program,
  - the demonstration of a sample solution of the program,
  - a benchmark of the sequential and of the parallel program in the style of Exercise 1.
- the source (`.java`) files of the sequential program and of the parallel program.

Exercise 3: Multi-Threaded/Network Programming in Java

You may use for this course the standard installation of Java (Java 6, `module load jdk`) or some more recent version (see `module avail` for all installed Java versions).

The goal of this exercise is to develop a multi-threaded client/server version of the Gaussian Elimination program developed in Exercise 2 in the programming language Java using Java's concurrency and networking API.

First, create a sequential Java solution for the problem. Demonstrate the correctness of your solution as in Exercise 2 and benchmark it with appropriate values for N (adjust the number of iterations in `smult` to get timings in a reasonable range).

Next, develop a multi-threaded version of the program. Use the high-level concurrency API to manage a fixed size pool of T threads for the multiple iterations of the algorithm (and simply generate as many `Callable` instances as is natural for the parallel execution of your program).

Write the program such that it can be started in one of two ways:

1. With the command line parameter `-server`: in this case the program is executed as a server which repeatedly waits (on some designated port) for the request of a client to create a random equation system of dimension N with seed R for the random number generator and solve the system with T threads; the server sends back to the client the number M of milliseconds that the solution of the equation system took.

2. With the command line parameter `-client` N R T: in this case, the program is started as a client that contacts the server on the designated port, sends the parameters N, R, and T to the server, waits for the result M, and prints M to the standard output.

Both server and clients may be run on the SGI UV machine. Please note that for a Java solution you may use the programs `PathThreadPool.java` and `MatMultNet.java` posted on the course site as a pattern.

For generating random numbers, use the class `java.util.Random`[1] of the Java standard library. For instance, assuming the declaration `import java.util.*;` the code

```
Random r = new Random(R);
for (int i=0; i<100; i++)
  System.out.println(r.nextDouble());
```

prints 100 floating point numbers generated by a random number generator with seed R.

For benchmarking Java programs, you may use the function

```
System.currentTimeMillis()
```

which returns the current wall clock time in milliseconds. Report the results as in Exercise 2 (state the version of Java that you used).

Bonus (30%): Rather than using floating point arithmetic, implement the program with arbitrary precision rational arithmetic. You may use for this the class `Rational` of the JScience library[2]. Use the command line option `-cp jscience.jar` for compiling and running your Java program with that library. In case the program runs much too fast for reasonable values of N, use a rational number equivalent of `smult`.

[1] https://docs.oracle.com/javase/8/docs/api/java/util/Random.html
[2] See http://jscience.org, sections "API" and "Download".